CONTENTS

5 Modelling and Model Fitting

5.1 Matrix and Vector Operations

5.1.1 Vector

Definition 1. A column of real numbers is called a **vector**.

Example 1.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix} \quad \mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

```
Python-code:
import numpy as np
a = np.array([[1],[-3],[2]])
print("a=",a)
One5 = np.ones(5).reshape(-1,1)
print("One5=",One5)
Output:
```

```
a= [[ 1]  [-3]   [ 2]]
One5= [[1.]  [1.]  [1.] [1.]  [1.]]
```

Since $\mathbf{y}$ has n elements it is said to have **order** (or dimension) n.

5.1.2 Matrix

Definition 2.
A rectangular array of elements with m rows and k columns is called an $m \times k$ **matrix.**

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix}$$

This matrix is said to be of **order** (or dimension) $m \times k$, where

- m is the **row** order (dimension)
- k is the **column** order (dimension)

- a_{ij} is the (i, j) element of $\mathbf{A}$.

Example 2.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & -2 \\ 0 & 4 & 5 \end{bmatrix} \quad \mathbf{I_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{J_3} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

```
Python-code:
import numpy as np
A = np.array([[1, 3, -2], [0, 4, 5]])
I = np.eye(3)
J = np.ones((3,3))
print("A=", A)
print("I3=",I)
print("J3=",J)
output:
A= [[ 1  3 -2]
 [ 0  4  5]]
I3= [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
J3= [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

5.1.3 Matrix Addition and Subtraction

Definition 3. Matrix addition
If $\mathbf{A}$ and $\mathbf{B}$ are both $m \times k$ matrices, then

$$\begin{aligned}\mathbf{C} &= \mathbf{A} + \mathbf{B} \\ &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{2k} \\ \vdots & \vdots & & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mk} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1k}+b_{1k} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2k}+b_{2k} \\ \vdots & \vdots & & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mk}+b_{mk} \end{bmatrix}\end{aligned}$$

Notation:

$$C_{m\times k} = \{c_{ij}\} \text{ where } c_{ij} = a_{ij} + b_{ij}$$

Definition 4. Matrix subtraction

If $\mathbf{A}$ and $\mathbf{B}$ are $m \times k$ matrices, then $\mathbf{C} = \mathbf{A} - \mathbf{B}$ is defined by

$$\mathbf{C} = \{c_{ij}\} \text{ where } c_{ij} = a_{ij} - b_{ij} \ .$$

Example 3.

$$\begin{bmatrix} 3 & 6 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 7 & -4 \\ -3 & 2 \end{bmatrix} = \begin{bmatrix} 10 & 2 \\ -1 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & -1 \\ 2 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ -1 & 1 \\ 0 & -1 \end{bmatrix}$$

```
Python-code:
import numpy as np
A1 = np.array([[3, 6], [2,1]])
A2= np.array([[7, -4], [-3, 2]])
A3 = A1 + A2
print("A1+A2=",A3)

output:
A1+A2= [[10  2]
 [-1  3]]
import numpy as np
B1 = np.array([[1, -1], [1,1],[1,0]])
B2= np.array([[1, -1], [2, 0],[1,1]])
B3 = B1 - B2
print("B1-B2=",B3)

Output:
B1-B2= [[ 0  0]
 [-1  1]
 [ 0 -1]]
```

5.1.4 Scalar Multiplication

Definition 5. Scalar multiplication

Let a be a scalar and $\mathbf{B} = \{b_{ij}\}$ be an $m \times k$ matrix, then

$$a\,\mathbf{B} = \mathbf{B}\,a = \{a\,b_{ij}\}$$

Example 4.

$$2\begin{bmatrix} 2 & -1 & 3 \\ 0 & 4 & -2 \end{bmatrix} = \begin{bmatrix} 4 & -2 & 6 \\ 0 & 8 & -4 \end{bmatrix}$$

```
 Python-Code:
import numpy as np
B1 = np.array([[2, -1, 3], [0,4,-2]])
B2 = 2*B1
print("2*B1=",B2)
Output:
B2=2*B1= [[ 4 -2  6]
          [ 0  8 -4]]
```

5.1.5 Transpose

Definition 6. Transpose
The transpose of the $m \times k$ matrix $\mathbf{A} = \{a_{ij}\}$ is the $k \times m$ matrix with elements $\{a_{ji}\}$. The transpose of $\mathbf{A}$ is denoted by $\mathbf{A^T}$ (or $\mathbf{A'}$).

Example 5.

$$\mathbf{A} = \begin{bmatrix} 1 & 4 \\ 3 & 0 \\ -2 & 6 \end{bmatrix} \qquad \mathbf{A^T} = \begin{bmatrix} 1 & 3 & -2 \\ 4 & 0 & 6 \end{bmatrix}$$

```
R-code:
import numpy as np
B1 = np.array([[1,4], [3,0],[-2,6]])
B2 = B1.T
print("B1^T = ",B2)

Output:
B1^T =
[[ 1  3 -2]
 [ 4  0  6]]
```

5.1.6 Inner Product

Definition 7. Inner product (crossproduct) of two vectors of order n

$$\mathbf{a}^T\mathbf{y} = [a_1, a_2, \cdots a_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = a_1y_1 + a_2y_2 + \cdots + a_ny_n$$

Example 6.

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 11 \\ 15 \\ 23 \end{bmatrix}$$

$$\mathbf{a^T y} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 11 \\ 15 \\ 23 \end{bmatrix} = 1(11) + 2(15) + 3(23) = 110$$

```
Python-codes:
import numpy as np
a = np.array([[1], [2], [3]])
y = np.array([[11], [15], [23]])
aty = a.T@y
print("a^Ty=",aty)
Output:
a^Ty= [[110]]
```

5.1.7 Matrix Multiplication

Definition 8. Matrix multiplication
The product of an $n \times k$ matrix $\mathbf{A}$ and a $k \times m$ matrix $\mathbf{B}$ is the $n \times m$ matrix $\mathbf{C} = \{c_{ij}\}$ with elements

$$c_{ij} = a_{i1}\, b_{1j} + a_{i2}\, b_{2j} + \cdots + a_{ik}\, b_{kj}$$

Example 7.

$$\mathbf{A} = \begin{bmatrix} 3 & 0 & -2 \\ 1 & -1 & 4 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \quad \mathbf{C} = \mathbf{AB} = \begin{bmatrix} 1 & -3 \\ 4 & 11 \end{bmatrix}$$

```
Python-codes:
import numpy as np
A = np.array([[3, 0, -2], [1, -1, 4]])
B = np.array([[1,1],[1, 2],[1,3]])
C = A@B
print("C=AB",C)
Output:
C=AB=
[[ 1 -3]
 [ 4 11]]
```

Example 8.

Consider the following matrix:

$$\mathbf{A} = \begin{bmatrix} 16.9 & 15.9 & 14.9 \\ 12.6 & 11.6 & 13.6 \\ 12.5 & 15.5 & 10.5 \end{bmatrix}.$$

Use Python to find the $(1, 3)$ element of $\mathbf{A^T A}$.

Sol:
$A^T A$ = [[600.62 608.62 554.42] [608.62 627.62 557.42] [554.42 557.42 517.22]]
$(1, 3)^{th}$ element of $\mathbf{A^T A}$ = 554.4200000000001

```
import numpy as np
A = np.array([[16.9,15.9,14.9],[12.6,11.6,13.6],
              [12.5,15.5,10.5]])
ATA = A.T@A
print("A^TA=",ATA)
ije = ATA[0,2]
print("(1,3) element of A = ", ije)
Output:
A^TA= [[[600.62 608.62 554.42]
 [608.62 627.62 557.42]
 [554.42 557.42 517.22]]]
(1,3) element of A =  554.4200000000001
```

5.1.8 Elementwise multiplication

Definition 9. Elementwise multiplication of two matrices

$$\mathbf{A} \# \mathbf{B} = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{k1} & \cdots & a_{km} \end{bmatrix} \# \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & & \vdots \\ b_{k1} & \cdots & b_{km} \end{bmatrix} = \begin{bmatrix} a_{11}\,b_{11} & \cdots & a_{1m}\,b_{1m} \\ \vdots & & \vdots \\ a_{k1}\,b_{k1} & \cdots & a_{km}\,b_{km} \end{bmatrix}$$

Example 9.

$$\begin{bmatrix} 3 & 1 \\ 2 & 4 \\ 0 & 6 \end{bmatrix} \# \begin{bmatrix} 1 & -5 \\ -3 & 4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 3 & -5 \\ -6 & 16 \\ 0 & 12 \end{bmatrix}$$

```
Python-codes:
import numpy as np
A = np.array([[3, 1], [2, 4],[0,6]])
B = np.array([[1,-5],[-3, 4],[-2,2]])
C = A*B
print("C=AB=",C)
Output:
C=AB=
[[ 3 -5]
 [-6 16]
 [ 0 12]]
```

5.1.9 Kronecker Product

Definition 10. Kronecker product of two matrices

$$\mathbf{A}_{k\times m} \otimes \mathbf{B}_{n\times s} = \begin{bmatrix} a_{11}\,\mathbf{B} & a_{12}\,\mathbf{B} & \cdots & a_{1m}\,\mathbf{B} \\ a_{21}\,\mathbf{B} & a_{22}\,\mathbf{B} & \cdots & a_{2m}\,\mathbf{B} \\ \vdots & \vdots & & \vdots \\ a_{k1}\,\mathbf{B} & a_{k2}\,\mathbf{B} & \cdots & a_{km}\,\mathbf{B} \end{bmatrix}$$

Example 10.

$$\begin{bmatrix} 2 & 4 \\ 0 & -2 \\ 3 & -1 \end{bmatrix} \otimes \begin{bmatrix} 5 & 3 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 6 & 20 & 12 \\ 4 & 2 & 8 & 4 \\ 0 & 0 & -10 & -6 \\ 0 & 0 & -4 & -2 \\ 15 & 9 & -5 & -3 \\ 6 & 3 & -2 & -1 \end{bmatrix}$$

$$\mathbf{a} \otimes \mathbf{y} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \otimes \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_1\, y_1 \\ a_1\, y_2 \\ a_2\, y_1 \\ a_2\, y_2 \\ a_3\, y_1 \\ a_3\, y_2 \end{bmatrix}$$

```
import numpy as np
A = np.array([[2, 4], [0, -2],[3,-1]])
B = np.array([[5,3],[2, 1]])
C = np.kron(A,B)
print("C=AB =", C)
print(C)
Output:
C=AB =
[[ 10   6  20  12]
 [  4   2   8   4]
 [  0   0 -10  -6]
 [  0   0  -4  -2]
 [ 15   9  -5  -3]
```

5.1.10 Determinant

Definition 11. The **determinant** of an $n \times n$ matrix $\mathbf{A}$ is

$$|\mathbf{A}| = \sum_{j=1}^{n} a_{ij}(-1)^{i+j}|M_{ij}| \quad \text{for any row } i$$

or

$$|\mathbf{A}| = \sum_{i=1}^{n} a_{ij}(-1)^{i+j}|M_{ij}| \quad \text{for any column } j$$

where M_{ij} is the "minor" for a_{ij} obtained by deleting the i^{th} row and j^{th} column from A.

Example 11.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$|\mathbf{A}| = a_{11}(-1)^{1+1}|a_{22}| + a_{12}(-1)^{1+2}|a_{21}|$$

then $\begin{vmatrix} 7 & 2 \\ 4 & 5 \end{vmatrix} =$

Example 12.

$$|\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

$$= a_{11}(-1)^{1+1} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} + a_{12}(-1)^{1+2} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix}$$

$$+ a_{13}(-1)^{1+3} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

$$\text{then } \begin{vmatrix} 1 & 1 & 3 \\ 4 & 8 & 6 \\ 7 & 5 & 9 \end{vmatrix} =$$

```
Python Code:
import numpy as np
A = np.array([[7, 2], [4, 5]])
B = np.array([[1,1,3],[4, 8,6],[7,5,9]])
DA = np.linalg.det(A)
DB = np.linalg.det(B)
print("DA=|A|=", DA)
print("DB=|B|=",DB)
Output:
DA=|A|=  27.0
DB=|B|=  -59.999999999999986
```

5.1.11 Inverse

Definition 12. The **identity matrix**, denoted by $\mathbf{I}$, is a $k \times k$ matrix of the form

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Example 13. $\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

```
I4 = np.eye(4)
print(''I4='',I4)
```

Definition 13. The **inverse** of a square, non-singular matrix $\mathbf{A}$ is the matrix, denoted by $\mathbf{A}^{-1}$, such that

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = I$$

Example 14.

Determine the inverse of

$$\mathbf{A} = \begin{pmatrix} 7 & 2 \\ 4 & 5 \end{pmatrix} = \text{and } \mathbf{B} = \begin{pmatrix} 1 & 1 & 3 \\ 4 & 8 & 6 \\ 7 & 5 & 9 \end{pmatrix}$$

```
import numpy as np
A = np.array([[7, 2], [4, 5]])
B = np.array([[1,1,3],[4, 8,6],[7,5,9]])
IA = np.linalg.inv(A)
IB = np.linalg.inv(B)
print("IA=A^(-1)=", IA)
print("IB=B^(-1)=",IB)
Output:
IA=A^(-1)= [[ 0.18518519 -0.07407407]
 [-0.14814815  0.25925926]]
IB=B^(-1)= [[-0.7        -0.1         0.3       ]
 [-0.1         0.2        -0.1       ]
 [ 0.6        -0.03333333 -0.06666667]]
```

5.1.12 Eigenvalues and Eigenvectors

Definition 14. For a $k \times k$ matrix $\mathbf{A}$, the scalars $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$ satisfying the polynomial equation

$$|\mathbf{A} - \lambda\,\mathbf{I}| = 0$$

are called the eigenvalues (or characteristic roots) of $\mathbf{A}$.

Definition 15. Corresponding to any eigenvalue λ_i is an eigenvector (or characteristic vector) $\mathbf{u}_i \neq \mathbf{0}$ satisfying

$$\mathbf{A}\,\mathbf{u}_i = \lambda_i\,\mathbf{u}_i.$$

Example 15. Find the eigenvalue and eigenvector of

$$\mathbf{A} = \begin{bmatrix} 1.96 & 0.72 \\ 0.72 & 1.54 \end{bmatrix}$$

Sol:

```
 Python codes:
import numpy as np
from scipy import linalg
A = np.array([[1.96, .72], [.72,1.54]])
w, v = linalg.eig(A)
print("w=", w)
print("v=",v)
Output:
w=
[2.5 1. ]
v=
[[ 0.8 -0.6]
 [ 0.6  0.8]]
```

5.1.13 Covariance Matrix

The sample **covariance**

$$s_{ik} = \frac{1}{n-1}\sum_{j=1}^{n}(x_{ji} - \bar{x}_i)(x_{jk} - \bar{x}_k)$$
$$i = 1, 2, , \ldots, p, k = 1, 2, \ldots, p$$

measures the association between the i^{th} and the k^{th} variables.

Notes:

- When $i = k$, the covariance reduces to sample variance.
- $s_{ik} = s_{ki}$ for all i and k.

The covariances computed from n measurements on p variables can be organized into arrays.

Covariance: $\mathbf{S_n} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1p} \\ s_{21} & s_{22} & \cdots & s_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{p1} & s_{p2} & \cdots & s_{pp} \end{bmatrix}$

Example 16.
Consider the following data about 3 variables:

x_1	x_2	x_3
45	38	10
37	31	15
42	26	17
35	28	21
39	33	12

Derive the sample covariance matrix using the NumPy package.

Sol:

```
import numpy as np
x1 = [45, 37, 42, 35, 39]
x2 = [38, 31, 26, 28, 33]
x3 = [10, 15, 17, 21, 12]
data = np.array([x1, x2, x3])
Sn = np.cov(data, bias=False)
print(Sn)
Output:
[[ 15.8    9.6  -12.  ]
 [  9.6   21.7  -17.25]
 [-12.   -17.25  18.5 ]]
```

Example 17.

You are given the following data:

No.	x_1	x_2	x_3
1	24.3	2.4	113.7
2	12.7	1.2	55.6
3	13.2	1.2	57.8
4	10.3	0.9	43.6
5	20.4	2.0	94.0
6	3.9	0.2	11.7
7	25.0	2.5	116.9
8	11.9	1.1	51.7
9	17.6	1.7	79.8
10	14.6	1.4	65.0

Write down the Python codes to determine the determinant of the sample covariance matrix($\mathbf{S_n}$).

Sol:

$\mathbf{S_n}$ = [[4.27210000e+01 4.61177778e+00 2.13276444e+02] [4.61177778e+00 4.98222222e-01 2.30235556e+01] [2.13276444e+02 2.30235556e+01 1.06476400e

$|\mathbf{S_n}|$ = 0.00035294759945734704

```
import numpy as np
x1 = [24.3,12.7,13.2,10.3,20.4,3.9,25.0,11.9,17.6,14.6]
x2 = [2.4,1.2,1.2,0.9,2.0,0.2,2.5,1.1,1.7,1.4]
```

```
x3 = [113.7,55.6,57.8,43.6,94.0,11.7,116.9,51.7,79.8,65.0]
data = np.array([x1, x2, x3])
Sn = np.cov(data, bias=False)
DetSn = np.linalg.det(Sn)
print("Det(S_n) = ",DetSn)
Output:
Det(S_n) =  0.00035294759945734704
```

5.2 Python for Probability

5.2.1 Factorial

Compute $n!$ as an Integer. For example computes 6!.

```
Python-codes:
import math
x =  math.factorial(6)
print("x=",x)
Output:
x= 720
```

5.2.2 Choose

Compute $\binom{m}{n}$. For example compute $\binom{6}{4}$.

```
Python-codes:
import math
y =  math.comb(6,4)
print("y=",y)
Output:
y= 15
```

5.2.3 Distributions

In Probability and Statistics II, for a random variable, X, we are interested to calculate the coresponding probability and it's moments, especially the expected value and variance. In the scipy stats library, it defines all the functions we would care about for a random variable, (X), including probabilities, expectation and variance. we also can generates a few random samples from X

- **Binomial Distribution** We know that for $X \sim Bin(n, p)$, the pmf of X is

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

For example, $X \sim Bin(10, 0.2)$, obtain the followings from Python:

(a.) Declare X to be a binomial random variable.

```
from scipy import stats
X = stats.binom(10, 0.2)
# Declare X to be a binomial random variable
```

(b.) Find $P(X = 2)$.

```
from scipy import stats
X = stats.binom(10, 0.2)
# Declare X to be a binomial random variable
fx2 = X.pmf(2) #P(X = 2)
print("P(X = 2)=",fx2)
Output:
P(X = 2)= 0.30198988800000004
```

(c.) Find $P(X \leq 3)$.

```
from scipy import stats
X = stats.binom(10, 0.2)
# Declare X to be a binomial random variable
Fx3 = X.cdf(3) #P(X <= 3)
print("P(X <= 3)=",Fx3)
Output:
P(X <= 3)= 0.879126118399999
```

(d.) Find $E(X)$.

```
from scipy import stats
X = stats.binom(10, 0.2)
# Declare X to be a binomial random variable
EX = X.mean() #E(X)
print("E(X)=",EX)
Output:
E(X)= 2.0
```

(e.) Find $Var(X)$.

```
from scipy import stats
X = stats.binom(10, 0.2)
# Declare X to be a binomial random variable
VX = X.var() #Var(X)
print("Var(X)=",VX)
Output:
Var(X)= 1.6
```

(f.) Get a random sample from X.

```
from scipy import stats
X = stats.binom(10, 0.2)
# Declare X to be a binomial random variable
rs1 = X.rvs() #Get a random sample from X
print("Random sample=",rs1)
Output:
Random sample= 1
```

(g.) Get 5 random samples from X.

```
from scipy import stats
X = stats.binom(10, 0.2)
# Declare X to be a binomial random variable
rs5 = X.rvs(5) #Get a random sample from X
print("Random sampled =",rs5)
Output:
Random samples = [2 2 2 1 4]
```

- **Poisson** $Y \sim Poi(\lambda)$, $P(Y = k) = \frac{\lambda^k e^{-\lambda}}{k!}$. For example, if $Y \sim Poi(3)$.

(a.) Declare Y to be a Poisson random variable.

```
from scipy import stats
Y = stats.poisson(3)
# Declare Y to be a poisson random variable
```

(b.) Find $P(Y = 2)$.

```
from scipy import stats
Y = stats.poisson(3)
# Declare Y to be a poisson random variable
fy2 = Y.pmf(2) #P(Y = 2)
print("P(Y = 2)=",fy2)
Output:
P(Y = 2)= 0.2240418076553877 5
```

(c.) Find $P(X \leq 2)$.

```
from scipy import stats
Y = stats.poisson(3)
# Declare Y to be a poisson random variable
Fy2 = Y.cdf(2) #P(Y <= 2)
print("P(Y <= 2)=",Fy2)
Output:
P(Y <= 2)= 0.4231900811268436 4
```

(d.) Find $E(Y)$.

```
from scipy import stats
Y = stats.poisson(3)
# Declare X to be a poisson random variable
EY = Y.mean() #E(Y)
print("E(Y)=",EY)
Output:
E(Y)= 3.0
```

(e.) Find $Var(Y)$.

```
from scipy import stats
Y= stats.poisson(3)
# Declare X to be a poisson random variable
VY = Y.var() #Var(Y)
print("Var(Y)=",VY)
Output:
Var(Y)= 3.0
```

(f.) Get a random sample from Y.

```
from scipy import stats
Y = stats.poisson(3)
# Declare Y to be a poisson random variable
rs1 = Y.rvs() #Get a random sample from Y
print("Random sample=",rs1)
Output:
Random sample= 2
```

(g.) Get 5 random samples from Y.

```
from scipy import stats
Y = stats.poisson(3)
# Declare Y to be a poisson random variable
rs5 = Y.rvs(5) #Get a random sample from Y
print("Random sampled =",rs5)
Output:
Random samples = [11  1  0  2  3]
```

- Normal Distribution
  Let $A \sim N(\mu, \sigma^2)$. Then

$$f_A(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The second parameter to a normal was the variance (σ^2). In the scipy library, the second parameter is the standard deviation (σ). For example. let $A \sim N(\mu = 10, \sigma^2 = 9)$.

(a.) Declare Y to be a Normal random variable.

```
from scipy import stats
A = stats.norm(10, math.sqrt(9))
# Declare A to be a normal random variable
```

(b.) Find $f_A(2)$.

```
import math
from scipy import stats
A = stats.norm(10, math.sqrt(9))
# Declare A to be a normal random variable
fA2 = A.pdf(2) # f(2), the probability density
print("f(2)=",fA2)
Output:
f(2)= 0.00379866200793248 1
```

(c.) Find $F(8) = P(X \leq 8)$.

```
import math
from scipy import stats
A = stats.norm(10, math.sqrt(9))
# Declare A to be a normal random variable
FA8 = A.cdf(8) #P(Y <= 8) or F(8)
print("F(8) =",FA8)
Output:
F(8) = 0.252492537546922 9
```

(d.) Find q such that $F(q) = P(X \leq q) = 0.95$.

```
import math
from scipy import stats
A = stats.norm(10, math.sqrt(9))
# Declare A to be a normal random variable
q = A.ppf(.95) # $F(q) = P(X \le q) = 0.95$
print("q =",q)
```

```
Output:
q = 14.934560880854416
```

(e.) Get 5 random samples from A.

```
import math
from scipy import stats
A = stats.norm(10, math.sqrt(9))
# Declare A to be a normal random variable
rs5 = A.rvs(5) #Get a random sample from A
print("Random sampled =",rs5)
Output:
Random samples = 10.89277752  7.19422281  7.32
```

- T Distribution
  We usually need to calculate the critical values or the p-values in hypothesis testing. In probability and statistics II, we know that T distribution depends on it's degrees of freedom, v, thus $T \sim T(v)$.

(a) Declare T as a T random variable with 10 degrees of freedom.

```
from scipy import stats
T = stats.t(10)
# Declare T to be a andom variable
```

(b) Find $P(T \leq 2)$.

```
from scipy import stats
T = stats.t(10)
# Declare T to be a andom variable
Ft2 = T.cdf(2) #P(T <= 2)
print("P(T <= 2)=",Ft2)
Output:
P(T <= 2)= 0.963305982614629 7
```

(c) Find $P(T \geq 3)$.

```
from scipy import stats
T = stats.t(10)
# Declare T to be a andom variable
St3 = 1-T.cdf(3) #P(T >= 3)
print("P(T >= 3)=",St3)
Output:
P(T >= 3)= 0.006671827511284811
```

(d) Find t_1 such that $P(T \leq t_1) = 0.98$.

```
from scipy import stats
T = stats.t(10)
# Declare T to be a andom variable
t1 = T.ppf(.98) #$P(T \le t_1) = 0.98$
print("t1=",t1)
Output:
t1= 2.359314623683182
```

(e) Find t_2 such that $P(T \geq t_2) = 0.1$.

```
from scipy import stats
T = stats.t(10)
# Declare T to be a andom variable
t2 = T.ppf(0.9) #$P(T \ge t_2) = 0.1$
print("t2=",t2)
Output:
t2= 1.372183641102863
```

- F Distribution
  In probability and statistics II, we know that F distribution depends on it's numerastor degrees of freedom, v_1 and denomonator degrees of freedom, v_2, thus, $F \sim F(v_1, v_2)$. Let $t \sim T(10)$ and $F \sim F(2, 10)$,

(a) Declare F as a F random variable with 2 and 10 degrees of freedoms.

```
from scipy import stats
F = stats.f(2,10)
# Declare F to be a andom variable
```

(b) Find $P(F \leq 2)$.

```
from scipy import stats
F = stats.f(2,10)
# Declare F to be a andom variable
F2 = F.cdf(2) #P(F <= 2)
print("P(F <= 2)=",F2)
Output:
P(F <= 2)= 0.814065567918129 3
```

(c) Find $P(F \geq 1)$.

```
from scipy import stats
F = stats.f(2,10)
# Declare F to be a andom variable
Sf1 = 1-F.cdf(1) #P(F >= 1)
print("P(F >= 1)=",Sf1)
Output:
P(F >= 1)= 0.401877572016460 9
```

(d) Find f_1 such that $P(F \leq f_1) = 0.95$.

```
from scipy import stats
F = stats.f(2,10)
# Declare F to be a andom variable
f1 = F.ppf(.95) #$P(T \le f_1) = 0.95$
print("f1=",f1)
Output:
f1= 4.102821015130400 5
```

(e) Find f_2 such that $P(F \geq f_2) = 0.01$.

```
from scipy import stats
F = stats.f(2,10)
# Declare F to be a andom variable
f2 = F.ppf(0.99) #$P(F \ge f_2) = 0.01$
print("f2=",f2)
Output:
f2= 7.559432157547899
```

5.3 System of Linear Equations

A system of linear equations (or linear system) is a collection of one or more linear equations involving the same set of variables. For example,

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= -2 \\ -x + \frac{1}{2}y - z &= 0 \end{aligned}$$

is a system of three equations in the three variables x, y, z. A solution to a linear system is an assignment of values to the variables such that all the equations are simultaneously satisfied. A solution to the system above is given by $x = 1, y = -2, z = -2$ since it makes all three equations valid. The word "system" indicates that the equations are to be considered collectively, rather than individually.

A general system of m linear equations with n unknowns can be written as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m_1}x_1 + a_{m_2}x_2 + \cdots + a_{m_n}x_n &= b_m \end{aligned}$$

where $x_1, x_2, \ldots, x_n$ are the unknowns, $a_{11}, a_{12}, \ldots, a_{mn}$ are the coefficients of the system, and $b_1, b_2, \ldots, b_n$ are the constant terms.

5.3.1 Solving System of Linear Equations Using Matrix Inversion

The system of linear equations can be expressed in matrix equation of the form $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{x}$ is a column vector with n entries, and $\mathbf{b}$ is a column vector with m entries.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

If the matrix $\mathbf{A}$ is square (has m rows and $n = m$ columns) and has full rank (all m rows are independent), then the system has a unique solution given by

$$\mathbf{x} = A^{-1}\mathbf{b}$$

Example 18.

Solve the system of linear equations below using matrix inversion.

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= -2 \\ -x + \frac{1}{2}y - z &= 0 \end{aligned}$$

Sol: $\mathbf{A} = \begin{bmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & \frac{1}{2} & -1 \end{bmatrix}$, $\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$, and $\mathbf{b} = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$

$\mathbf{A}^{-1} = \begin{bmatrix} 0 & -.5 & -2 \\ 0.67 & 1.33 & 4.67 \\ 0.33 & 1.17 & 3.33 \end{bmatrix}$, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \begin{bmatrix} 0 & -.5 & -2 \\ 0.67 & 1.33 & 4.67 \\ 0.33 & 1.17 & 3.33 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ -2 \end{bmatrix}$

```
Python code:
import numpy as np
A = np.array([[3, 2,-1], [2,-2,4],[-1,1./2,-1]])
b = np.array([[1],[-2],[0]])
IA = np.linalg.inv(A)
print("A^(-1}=",IA)
x = IA@b
print("x=A^(-1)b",x)
Output:
A^(-1}= [[ 2.22044605e-16 -5.00000000e-01 -2.00000000e+00]
 [ 6.66666667e-01  1.33333333e+00  4.66666667e+00]
 [ 3.33333333e-01  1.16666667e+00  3.33333333e+00]]
x=A^(-1)b [[ 1.]
 [-2.]
 [-2.]]
```

Example 19.

Consider the following linear system:

$$\begin{aligned} 36w + 26x + 21y + 32z &= 54 \\ 33w + 17x + 49y + 33z &= 16 \\ 33w + 36x + 18y + 32z &= 88 \\ 32w + 34x + 52y + 65z &= 72 \end{aligned}$$

(a) Write the above system in the form $\mathbf{AX} = \mathbf{b}$.

(b) Provide the Python codes to solve the linear system using matrix inversion.

(c) Suppose the unique solution to part (a) is $\mathbf{X} = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -0.596 \\ 3.0916 \\ -0.4319 \\ 0.1294 \end{bmatrix}$, write down the Python codes to find $24w + 33x + 14y + 24z$ and calculate the corresponding value.

Sol:

(a) $$\begin{bmatrix} 36 & 26 & 21 & 32 \\ 33 & 17 & 49 & 33 \\ 33 & 36 & 18 & 32 \\ 32 & 34 & 52 & 65 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 54 \\ 16 \\ 88 \\ 72 \end{bmatrix}$$

(b)
```
import numpy as np
from scipy import linalg
A = np.array([[36,26,21,32],[33,17,49,33],
[33,36,18,32],[32,34,52,65]])
b = np.array([[54],[16],[88],[72]])
x = linalg.inv(A)@b
print("x=",x)
Output:
x = [[-0.596] [3.0916] [-0.4319] [0.1294] ]
```

(c) $24w+33x+14y+24z = 24(-0.596)+33(3.0916)+14(-0.4319)+24(0.1294) = 84.78$

```
c = np.array([24,33,14,24])
bnew = c@x
print("bnew = ",bnew)
Output:
bnew = [84.78]
```

5.3.2 Solving System of Linear Equations Using linalg's Solve Routine

The system of linear equations can be also solve using linalg's Routine in numpy package. Instead of solving $\mathbf{x} = \mathbf{A^{-1}b}$. We can obtained the solution by using the linalg's solve routine.

Example 20.

Solve the system of linear equations below using linalg's solve routine.

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= -2 \\ -x + \frac{1}{2}y - z &= 0 \end{aligned}$$

```
Python code:
import numpy as np
from scipy import linalg
A = np.array([[3, 2,-1], [2,-2,4],[-1,1./2,-1]])
b = np.array([[1],[-2],[0]])
x = linalg.solve(A,b)
print("x=",x)
Output:
x [[ 1.]
 [-2.]
 [-2.]]
```

5.4 Regression Analysis

5.4.1 What Is Regression?

Regression searches for relationships among variables. For example, you can observe the selling price of properties and try to understand how the area of living space on the properties, the appraised home value on the properties, and appraised land value of the properties, the selling price depend on the features, such as the area of living space on the property, the appraised home value on the property, and appraised land value of the property, and so on.

This is a regression problem where data related to each property represent one observation. The presumption is that the area of living space on the property, the appraised home value on the property, and appraised land value of the property are the independent features, while the selling price depends on them.

Generally, in regression analysis, you usually con-

sider some phenomenon of interest and have a number of observations. Each observation has two or more features. Following the assumption that (at least) one of the features depends on the others, you try to establish a relation among them. In other words, you need to find a function that maps some features or variables to others sufficiently well. The dependent features are called the **dependent variables, outputs, or responses**. The independent features are called the **independent variables, inputs, or predictors**.

Regression problems usually have one continuous and unbounded dependent variable. The inputs, however, can be continuous, discrete, or even categorical data such as gender, nationality, brand, and so on.

It is a common practice to denote the outputs with y and inputs with x. If there are two or more independent variables, they can be represented as the vector $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the

number of inputs.

5.4.2 When Do You Need Regression?

Typically, you need regression to answer whether and how some phenomenon influences the other or how several variables are related. For example, you can use it to determine if and to what extent the area of living space on the property, the appraised home value on the property, and appraised land value of the property impact selling price.

Regression is also useful when you want to forecast a response using a new set of predictors. For example, you could try to predict electricity consumption of a household for the next hour given the outdoor temperature, time of day, and number of residents in that household.

Regression is used in many different fields: economy, computer science, social sciences, and so on. Its importance rises every day with the availability of large amounts of data and increased aware-

ness of the practical value of data.

5.4.3 Problem Formulation

When implementing linear regression of some dependent variable y on the set of independent variables $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the number of predictors, you assume a linear relationship between y and x:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \epsilon.$$

This equation is the regression equation. $\beta_0, \beta_1, \ldots, \beta_r$ are the regression coefficients, and ϵ is the random error.

Linear regression calculates the estimators of the regression coefficients, denoted with $b_0, b_1, \ldots, b_r$. They define the estimated regression function $\hat{y} = b_0 + b_1 x_1 + \cdots + b_r x_r$. This function should capture the dependencies between the inputs and output sufficiently well.

The estimated or predicted response, $\hat{y}_i$, for each observation $i = 1, \ldots, n$, should be as close as

possible to the corresponding actual response y_i. The differences $y_i - \hat{y}_i$ for all observations $i = 1, \ldots, n$, are called the residuals or errors. Regression is about determining the best predicted weights, that is the weights corresponding to the smallest residuals.
To get the best weights, you usually minimize the sum of squared residuals(errors) (SSE) for all observations $i = 1, \ldots, n$, $\text{SSE} = \sum_i (y_i - \hat{y}_i)^2$. This approach is called the method of ordinary least squares(OLS).

5.4.4 Regression Performance

The variation of actual responses $y_i, i = 1, \ldots, n$ occurs partly due to the dependence on the predictors x_i. However, there is also an additional inherent variance of the output.
The coefficient of determination, denoted as R^2, tells you which amount of variation in y can be explained by the dependence on x using the particular regression model. Larger R^2 indicates a

better fit and means that the model can better explain the variation of the output with different inputs.
The value $R^2 = 1$ corresponds to $SSE = 0$, that is to the perfect fit since the values of predicted and actual responses fit completely to each other. Thus, the larger the R^2, the better fit the data to the model. However, R^2 increases as the number parameters increases. Thus we usually will use adjusted R^2 to choose the appropriate model. Another two quantity that use to make decision on model selection are Aikaike Information Criterion(AIC) and Bayesisn Information Criterion(BIC).

5.4.5 Simple Linear Regression

Simple or single-variate linear regression is the simplest case of linear regression with a single independent variable, $\mathbf{x} = x$.
When implementing simple linear regression, you typically start with a given set of input-output (x, y) pairs. These pairs are your observations.

The estimated regression function has the equation $y = \beta_0 + \beta_1 x$. Your goal is to calculate the optimal values of the predicted weights β_0 and β_1 that minimize Sum of Square of Error (SSE) and determine the estimated regression function. The value of β_0, also called the intercept, shows the point where the estimated regression line crosses the y axis. It is the value of the estimated response $y = 0$ for $x = 0$. The value of β_1 determines the slope of the estimated regression line.

5.4.6 Multiple Linear Regression

Multiple or multivariate linear regression is a case of linear regression with two or more independent variables.

If there are just two independent variables, the estimated regression function is $\hat{y} = b_0 + b_1 x_1 + b_2 x_2$. It represents a regression plane in a three-dimensional space. The goal of regression is to determine the values of the weights b_0, b_1 and b_2 such that this plane is as close as possible to the

actual responses and yield the minimal SSE.
The case of more than two independent variables is similar, but more general. The estimated regression function is $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots b_r x_r$ and there are $r + 1$ weights to be determined when the number of inputs is r.

5.4.7 Polynomial Regression

You can regard polynomial regression as a generalized case of linear regression. You assume the polynomial dependence between the output and inputs and, consequently, the polynomial estimated regression function.
In other words, in addition to linear terms like $b_1 x_1$, your regression function y can include nonlinear terms such as $b_2 x_1^2$, $b_1 x_1^3$, or even $b_4 x_1 x_2$, $b_5 x_1 x_3$ and so on.
The simplest example of polynomial regression has a single independent variable, and the estimated regression function is a polynomial of degree 2: $\hat{y} = b_0 + b_1 x + \beta_2 x^2$.

Keeping this in mind, compare the previous regression function with the function $\hat{y} = b_0 + b_1 x + b_2 x^2$ used for linear regression. They look very similar and are both linear functions of the unknowns b_0, b_1, and b_2. This is why you can solve the polynomial regression problem as a linear problem with the term x^2 regarded as an input variable.

In the case of two variables and the polynomial of degree 2, the regression function has this form: $\hat{y} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_1^2 + b_4 x_1 x_2 + b_5 x_2^2$. The procedure for solving the problem is identical to the previous case. You apply linear regression for five inputs: $x_1, x_2, x_1^2, x_1 x_2$, and x_2^2. What you get as the result of regression are the values of six weights which minimize SSE: b_0, b_1, b_2, b_3, b_4, and b_5.

5.5 Ordinary Least Squares Estimation

For the linear model with

$$\mathbf{y} = \mathbf{X}\,\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

we have

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1r} \\ X_{21} & X_{22} & \cdots & X_{2r} \\ \vdots & \vdots & & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{nr} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_r \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

and

$$\begin{aligned} y_i &= \boldsymbol{\beta}_1\,\mathbf{x}_{i1} + \boldsymbol{\beta}_2\,\mathbf{x}_{i2} + \cdots + \boldsymbol{\beta}_r\,\mathbf{x}_{ir} + \boldsymbol{\epsilon}_i \\ &= \mathbf{X}_i^T\,\boldsymbol{\beta} + \boldsymbol{\epsilon}_i \end{aligned}$$

where $\mathbf{X}_i^T = (\mathbf{x}_{i1}, \mathbf{x}_{i2}, \cdots, \mathbf{x}_{ir})$ is the i-th row of the model matrix $\mathbf{X}$.

Definition 16.
For a linear model with $\mathbf{y} = \mathbf{X}\,\boldsymbol{\beta} + \boldsymbol{\epsilon}$ any vector $\mathbf{b}$ that minimizes the sum of squared residuals

$$\begin{aligned} Q(\mathbf{b}) &= \textstyle\sum_{i=1}^{n} (y_i - \mathbf{X}_i^T\mathbf{b})^2 \\ &= (\mathbf{y} - X\mathbf{b})^T(\mathbf{y} - \mathbf{X}\mathbf{b}) \end{aligned}$$

is an ordinary least squares (OLS) estimator for $\boldsymbol{\beta}$.

For $j = 1, 2, \ldots, r$, solve

$$0 = \frac{\partial Q(\mathbf{b})}{\partial b_j} = 2 \sum_{i=1}^{n} (y_i - \mathbf{X}_i^T \mathbf{b}) \, X_{ij}$$

Dividing by 2, we have

$$0 = \sum_{i=1}^{n} (y_i - \mathbf{X}_i^T \mathbf{b}) \, X_{ij} \quad j = 1, 2, \ldots, r$$

These equations are expressed in matrix form as

$$\begin{aligned} \mathbf{0} &= \mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{b}) \\ &= \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{b} \end{aligned}$$

or

$$\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y}$$

These are often called the "normal" equations.

If $\mathbf{X}_{n \times r}$ has full column rank, i.e., $\text{rank}(\mathbf{X}) = k$, then

$$(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X})\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

and

$$\boxed{\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$

is the unique solution to the normal equations.

If $\text{rank}(\mathbf{X}) < k$, then the solution does not exist.

Example 21. Regression Analysis:Yield of a chemical process

Yield (%)	Temperature (^oF)	Time (hr)
y	x_1	x_2
77	160	1
82	165	3
84	165	2
89	170	1
94	175	2

Multiple linear regression model

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \epsilon_i$$

$$i = 1, 2, 3, 4, 5$$

Matrix formulation:

$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} \\ \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} \\ \beta_0 + \beta_1 x_{31} + \beta_2 x_{32} \\ \beta_0 + \beta_1 x_{41} + \beta_2 x_{42} \\ \beta_0 + \beta_1 x_{51} + \beta_2 x_{52} \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & x_{11} & x_{21} \\ 1 & x_{12} & x_{22} \\ 1 & x_{13} & x_{23} \\ 1 & x_{14} & x_{24} \\ 1 & x_{15} & x_{25} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \end{bmatrix}$$

$$\begin{bmatrix} 77 \\ 82 \\ 84 \\ 89 \\ 94 \end{bmatrix} = \begin{bmatrix} 1 & 160 & 1 \\ 1 & 165 & 3 \\ 1 & 165 & 2 \\ 1 & 170 & 1 \\ 1 & 175 & 2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \end{bmatrix}$$

Example 22. Refer to example 21, obtain the OLS estimate, $\mathbf{b}$.

$$\mathbf{y} = \begin{bmatrix} 77 \\ 82 \\ 84 \\ 89 \\ 94 \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & 160 & 1 \\ 1 & 165 & 3 \\ 1 & 165 & 2 \\ 1 & 170 & 1 \\ 1 & 175 & 2 \end{bmatrix}, \text{ then}$$

$$\mathbf{X^T X} = \begin{bmatrix} 5 & 835 & 9 \\ 835 & 139,575 & 1,505 \\ 9 & 1,505 & 19 \end{bmatrix}, \mathbf{X^T y} = \begin{bmatrix} 426 \\ 71,290 \\ 768 \end{bmatrix}$$

$$\mathbf{X^T X^{-1}} = \begin{bmatrix} 2.149 & -1.289 & 0.278 \\ -1.29 & 0.0078 & -0.0056 \\ 0.278 & -0.0056 & 0.361 \end{bmatrix} \mathbf{X^T y} = \begin{bmatrix} 426 \\ 71,290 \\ 768 \end{bmatrix}$$

$$\begin{aligned} &\mathbf{b} \\ &= \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} \\ &= \mathbf{X^T X^{-1} X^T y} \\ &= \begin{bmatrix} 2.149 & -1.289 & 0.278 \\ -1.29 & 0.0078 & -0.0056 \\ 0.278 & -0.0056 & 0.361 \end{bmatrix} \begin{bmatrix} 426 \\ 71,290 \\ 768 \end{bmatrix} \\ &= \begin{bmatrix} -105.22 \\ 1.144 \\ -0.389 \end{bmatrix} \end{aligned}$$

Python code:

```
import numpy as np
y = [77,82,84,89,94]
x0 = np.ones(5).reshape(-1,1)
x1 = np.array([160,165,165,170,175]).reshape(-1,1)
x2 = np.array([1,3,2,1,2]).reshape(-1,1)
x = np.hstack((x0, x1,x2))
xtx = x.T@x
xty = x.T@y
b = np.linalg.inv(xtx)@xty
print("b=",b)
```

Output:

```
b= [-105.22222222    1.14444444   -0.38888889]
```

Example 23.

A researcher in a scientific foundation wished to evaluate the relation between intermediate and senior level annual salaries of bachelor's and master's level mathematician (Y , in thousand dollars) and index of work quality (X_1), number of years of experience (X_2). You fit the data below to $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$.

y	x_1	x_2
146	7.9	25.8
144	4.6	29.4
117	4.7	8.6
143	3.9	46.4
127	6.8	12.1
108	2.0	21.6
133	8.1	35.0
138	4.4	17.5
139	6.0	19.4
117	5.1	20.9

You are given that:

- n is the number of observations.
- p is the number of parameters in the model.
- $\widehat{\boldsymbol{\beta}} = \begin{bmatrix} \hat{\beta}_0 & \hat{\beta}_1 & \hat{\beta}_2 \end{bmatrix}$
- $SSR = \widehat{\boldsymbol{\beta}}^T \mathbf{X^T y} - \frac{1}{n} \mathbf{y^T J y}$, where $\mathbf{J}$ is an $n \times n$ matrix of one.

- $SSE = \mathbf{y^T y} - \widehat{\boldsymbol{\beta}}^T \mathbf{X^T y}$.
- $SST = \mathbf{y^T y} - \frac{1}{n}\mathbf{y^T J y}$.
- $MSE = \frac{SSE}{n-p}$
- $SE(\hat{\beta}_j) = \sqrt{MSE \times C_{jj}}$, where C_{jj} is the diagonal element of the $(\mathbf{X^T X})^{-1}$ corresponding to $\hat{\beta}_j$.

(a) Write the Python commands and output using matrix formulation to obtain the estimate of $\boldsymbol{\beta} = \begin{bmatrix}\beta_0 & \beta_1 & \beta_2\end{bmatrix}$.

(b) Determine the predicted value for the mean score of Y with $X_1 = [5.9]$ and $X_2 = [30.8]$.

(c) Write the Python commands and outputs to calculated SSR.

(d) Write the Python commands and outputs to calculated SSE.

(e) Write the Python commands and outputs to calculated SST.

(f) You are given that $R^2 = \frac{SSR}{SST}$ and adjusted R^2, $R^2_{Adj} = 1 - \frac{SSE/(n-p)}{SST/(n-1)}$. Write the Python commands and outputs to calculated R^2 and adjusted R^2.

(g) Suppose you are interested to test whether the number of years of experience (X_2) affect salaries, your hypotheses are $H_0 : \beta_2 = 0$ versus $H_1 : \beta_2 > 0$. The

corresponding test statistic for testing these hypotheses is $t = \frac{\hat{\beta}_2}{SE(\hat{\beta}_2)}$. Write the Python commands and outputs to calculated t.

(h) Write the Python commands and outputs to calculated the p-value of the test in part(g).

(i) Write the Python commands and outputs to calculated the test statistic for testing the hypothetses $H_0 : \beta_1 = \beta_2 = 0$ versus H_1 : At least one of the β's $\neq 0$.

(j) Write the Python commands and outputs to calculated the p-value of the test in part(i).

Sol:

(a) $\widehat{\boldsymbol{\beta}}$

$= \begin{bmatrix} \hat{\beta}_0 & \hat{\beta}_1 & \hat{\beta}_2 \end{bmatrix}$

$= \begin{bmatrix} 100.85505860082276 & 3.049850937364648 & 0.5926590149673068 \end{bmatrix}$

```
import numpy as np
y = [146.0, 144.0, 117.0, 143.0, 127.0,
108.0, 133.0, 138.0, 139.0,117.0]
n = len(y)
x0 = np.array(np.repeat(1,n)).reshape(-1,1)
x1 = np.array([7.9, 4.6, 4.7, 3.9, 6.8,
2.0, 8.1, 4.4, 6.0, 5.1]).reshape(-1,1)
x2 = np.array([25.8, 29.4, 8.6, 46.4, 12.1,
21.6, 35.0, 17.5, 19.4, 20.9]).reshape(-1,1)
x = np.hstack((x0, x1,x2))
```

```
x, y = np.array(x), np.array(y)
p = 3
xtx = x.T@x
xty = x.T@y
ixtx = np.linalg.inv(xtx)
b = ixtx@xty
print("b =", b)
Output:
b = [100.85505860082276 3.049850937364648 0.5926590149673
```

(b) $\hat{Y} = 100.8551 + 3.0499X_1 + 0.5927X_2 = 100.8551 + 3.0499([5.9]) + 0.5927([30.8]) = [137.1031]$

```
xh = np.array([[1],[5.9], [30.8]])
yh = xh.T@b
print("y hat=", yh)
Output:
y hat= [137.1031]
```

(c) $SSR = 697.926290239353$

```
bxty = b.T@xty
J = np.ones((n,n))
ytJy = y.T@J@y
SSR = bxty-ytJy/n
print("SSR=", SSR)

Output:
SSR= 697.926290239353
```

(d) `yty = y.T@y`

```
bxty = b.T@xty
SSE = yty-bxty
print("SSE=", SSE)
Output:
SSE= 893.6737097606529
```

(e)
```
yty = y.T@y
J = np.ones((n,n))
ytJy = y.T@J@y
SST = yty-ytJy/n
print("SST=",SST)

Output:
SST= 1591.6000000000058
```

(f)
```
RSq = SSR/SST
AdjRSq = 1-(SSE/(n-p))/(SST/(n-1))
print("Rsq=",RSq)
print("AdjRsq=", AdjRSq)

Output:
RSq= 0.4385060883635024
AdjRSq= 0.2780792564673602
```

(g)
```
import numpy as np
from numpy import sqrt,linalg
n = len(y)
p = 3
MSE = SSE/(n-p)
Cjj = ixtx[2,2]
SEbeta = sqrt(MSE*Cjj)
```

```
print("SEbeta = ", SEbeta)
t = b[2]/SEbeta
print("t = ",t)
Output:
SEbeta = 0.338937558704337
t = 1.7485787566089623
```

(h)
```
from scipy import stats
T = stats.t(n-p)
pvalue = 1-T.cdf(t)
print("p-value = ", pvalue)
Output:
p-value = 0.061924782029449 62
```

(i)
```
MSR = SSR/p
F = MSR/MSE
print("F = ", F)
Output:
F = 1.8222474930601982
```

(j)
```
FD = stats.f(p,n-p)
pv2 = 1-FD.cdf(F)
print("p-value = ", pv2)
Output:
p-value = 0.23093715578424823
```

5.6 Analysis of Variance(ANOVA)

In ANOVA, we wish to compare the mean responses for several populations where the levels of a single explanatory variable define the populations. That is we want to test:
$H_0 : \mu_1 = \mu_2 = \mu_3 = \cdots = \mu_k$
H_1 : Not all population means are equal.

The basic concept underlying ANOVA:
To assess whether the population means are equal, the variation between the sample means is compared to the natural variation of the observations within the samples via their ratio that is called the F-statistic. The larger the variation between the sample means compared to the natural variation within the samples—that is, the larger the value of the F-statistic—the more support there is of a difference in the population means
The between variability will be contrasted to how much natural variation there is within the groups.
The test-statistic actually used to make the decision is called an F-statistic and can be loosely viewed as follows:

$$F = \frac{\text{Variation BETWEEN the sample means}}{\text{Natural Variation WITHIN the samples}} = \frac{MST}{MSE},$$

where $MST = \frac{SST}{k-1}$ and $MSE = \frac{SSE}{N-k}$, where $N = n_1 + n_2 + \cdots + n + k$.

Notation in ANOVA(k = Number of populations or groups under study)

	Population 1	Population 2	$\cdots$	Population k
Data	$y_{11}, y_{12}, \ldots, y_{1n_1}$	$y_{21}, y_{22}, \ldots, y_{2n_2}$	$\cdots$	$y_{k1}, y_{k2}, \ldots, y_{kn_k}$
Means	μ_1	μ_2	$\cdots$	μ_k

The data can be model as $y_{ij} = \mu_i + \epsilon_{ij}$.
We can express this model in matrix form as $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$

$$\underset{\uparrow\ \mathbf{y}}{\begin{bmatrix} y_{11} \\ y_{12} \\ \vdots \\ y_{1n_1} \\ y_{21} \\ y_{22} \\ \vdots \\ y_{2n_2} \\ \vdots \\ y_{k1} \\ y_{k2} \\ \vdots \\ y_{kn_k} \end{bmatrix}} = \underset{\uparrow\ \mathbf{X}}{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}} \underset{\uparrow\ \boldsymbol{\beta}}{\begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \vdots \\ \mu_k \end{bmatrix}} + \underset{\uparrow\ \boldsymbol{\epsilon}}{\begin{bmatrix} \epsilon_{11} \\ \epsilon_{12} \\ \vdots \\ \epsilon_{1n_1} \\ \epsilon_{21} \\ \epsilon_{22} \\ \vdots \\ \epsilon_{2n_2} \\ \vdots \\ \epsilon_{k1} \\ \epsilon_{k2} \\ \vdots \\ \epsilon_{kn_k} \end{bmatrix}}$$

Furthermore, if $n_i = n$, then $\mathbf{X}$ can be expressed in kronecker form as $\mathbf{I_k} \otimes \mathbf{1_n}$.

The estmate of β is $\mathbf{b} = (\mathbf{X^T X})^{-1}\mathbf{X^T y}$

The null hypothesis $H_0 : \mu_1 = \mu_2 = \mu_3 = \cdots = \mu_k$ can be expresed as

$$H_0 : \begin{bmatrix} \mu_1 - \mu_k = 0 \\ \mu_2 - \mu_k = 0 \\ \vdots \\ \mu_{k-1} - \mu_k = 0 \end{bmatrix}$$

$$H_0 : \begin{bmatrix} 1 & 0 & \cdots & 0 & -1 \\ 0 & 1 & \cdots & 0 & -1 \\ \vdots & & & & \\ 0 & 0 & \cdots & 1 & -1 \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad H_0 : \mathbf{C}\boldsymbol{\beta} = \mathbf{0}$$

Thus, $\mathbf{C} = [\mathbf{I}_{k-1} | -\mathbf{1}_{k-1}]$

Next,

$SST = (\mathbf{Cb})^T[\mathbf{C}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{C}^T]^{-1}(\mathbf{Cb})$.

To find SSE, we first obtain the error, $\mathbf{e} = \mathbf{y} - \mathbf{Xb}$, and $SSE = \mathbf{e^T e}$.

Finally, we find $F = \frac{MST}{MSE} = \frac{SST/k-1}{SSE/N-k}$

We reject H_0 if

$F > F_{k-1,N-k,\alpha}$ or

p-value $= P(F_{k-1,N-k} \geq F) < \alpha$

Example 24.

An experiment was run to determine whether four specific firing temperatures affect the density of a certain type of brick. The experiment led to the following data.

Temperature	Density				
100	21.8	21.9	21.7	21.6	21.7
125	21.7	21.4	21.5	21.4	21.6
150	21.9	21.8	21.8	21.6	21.5
175	21.9	21.7	21.8	21.4	21.3

Consider the model $y_{ij} = \mu_i + \epsilon_{ij}, i = 1, 2, 3, 4; j = 1, 2, \ldots, 5,$ where

- y_{ij} is the density of the j^{th} brick with i^{th} firing temperature.
- μ_i is the mean density of brick with i firing temperature.
- $\epsilon_{ij} \sim N(0, \sigma^2)$

Use this model to answer the following questions.

(a) Let $\boldsymbol{\beta} = (\mu_1, \mu_2, \mu_3, \mu_4)^T$, $\mathbf{y} = [y_{11}, y_{12}, y_{13}, y_{14}, y_{15}, y_{21}, y_{22}, y_2$ and $\boldsymbol{\epsilon} = [\epsilon_{11}, \epsilon_{12}, \epsilon_{13}, \epsilon_{14}, \epsilon_{15}, \epsilon_{21}, \epsilon_{22}, \epsilon_{23}, \epsilon_{24}, \epsilon_{25}, \epsilon_{31}, \epsilon_{32}, \epsilon_{33}, \epsilon_{34}$ The above model can be express in the form $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$. Write down the matrix of $\mathbf{X}$ in kronecker form.

(b) Write down the Python codes and the corresponding output to obtain the estimate of $\boldsymbol{\beta}$, $\mathbf{b}$.

(c) The hypotheses to test the equalities of means are H_0 : $\mu_1 = \mu_2 = \mu_3 = \mu_4$ versus H_1: at least one population mean is different from the rest. H_0 can be express as H_0 : $\mathbf{C}\boldsymbol{\beta} = \mathbf{d}$. Determine a matrix $\mathbf{C}$ so that H_0 : $\mathbf{C}\boldsymbol{\beta} = \mathbf{0}$.

(d) Continue using the Python codes from part (b), write down the Python codes and the corresponding output to compute the sum of squares of firing temperature, SST.

(e) Continue using the Python codes from part (b), write down the Python codes and the corresponding output to compute the sum of squares of error, SSE.

(f) Continue using the Python codes from part (b), (d) and (e), write down the Python codes and the corresponding output to compute the test statistic.

(g) Write down the Python codes and the corresponding output to compute the p-value

Sol:

(a) $\mathbf{X} = \mathbf{I}_4 \otimes \mathbf{1}_5$

(b)
```
import numpy as np
y1 = [21.8,21.9,21.7,21.6,21.7]
y2 = [21.7,21.4,21.5,21.4,21.6]
y3 = [21.9,21.8,21.8,21.6,21.5]
```

```
y4 = [21.9,21.7,21.8,21.4,21.3]
y = np.hstack((y1,y2,y3, y4))
I4 = np.eye(4)
One5 = np.ones(5).reshape(-1,1)
x = np.kron(I4,One5)
xtx = x.T@x
ixtx = np.linalg.inv(xtx)
xty = x.T@y
b = ixtx@xty
print("b=",b)
Output:
b= [21.74 21.52 21.72 21.62]
```

(c) $\mathbf{C} = [\mathbf{I_3}| - \mathbf{1_3}] = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix}$

(d)
```
I3 = np.eye(3)
One3 = np.ones(3).reshape(-1,1)
C = np.hstack((I3,-One3))
print("C=",C)
Cb = C@b
SST = Cb.T@np.linalg.inv((C@ixtx@C.T)) @Cb
print("SST",SST)
Output:
SST 0.1540000000000023
```

(e)
```
e = y-x@b
```

```
print(e)
SSE = e.T@e
print("SSE=",SSE)
```

Output: SSE= 0.4959999999999989

(f)
```
N = 20
k = 4
F = (SST/k)/(SSE/(N-k))
print("F=",F)
Output:
F= 1.241935483870989
```

(g)
```
from scipy import stats
FD = stats.f(k,N-k)
pvalue = 1-FD.cdf(F)
print("p-value=",pvalue)
Output:
p-value= 0.332853264899390 4
```